Apart from using Trie on words we can also use it on numbers.
The most common use case is find a maximum XOR between a certain number and list of numbers.

The most important intuition to understand this problem is how XOR works.
When we XOR two bits(0 or 1) it results in 1 if both bits have different values. In other words XOR of 'N' odd count of 1s OR 0s results in 1 or else 0.

Now, when we want max value of suppose one byte 00000001(1) and a list of bytes. To find the max we ideally need the exact opposite of above bits.

Like 1 1 1 1 1 1 1 0, XORing them would lead to all 1s leading to max value and we consider leftmost (Most Significant Bits) first because they have most value.

To create Trie of number our logic is same as normal Trie with small differences.
- We only needs two elements in link[] array for 0 & 1.

```
class Trie {
    private Node root;

    Trie() {
        this.root = new Node();
    }

    public void insert(int num) {
        Node node = root;
        for (int i = 31; i >= 0; i--) {
            int bit = (num >> i) & 1;
            if (!node.containsKey(bit))
                node.put(bit);
            node = node.get(bit);
        }
    }

    public int getMax(int num) {
        Node node = root;
        int maxNum = 0;
        for (int i = 31; i >= 0; i--) {
            int bit = (num >> i) & 1;
            if (node.containsKey(1 - bit)) {
                maxNum = maxNum | (1 << i);
                node = node.get(1 - bit);
            } else
                node = node.get(bit);
        }
        return maxNum;
    }
}
```

We can see the helper Trie methods which uses Node class & it's helper methods.

Insert method goes from 32nd bit to 1st bit position wise and updates trie with new nodes.

-> If we have the complement of current position bit that means we can use this bits value and by doing OR with 1 at that position we make sure that this bit is always 1.